САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра системного программирования

Афанасенко Никита Владимирович

Разработка универсальной платформы для идентификации объектов по минимально возможному числу признаков

Выпускная работа бакалавра

Допущена к защите.

Зав. Кафедрой:
д. ф.-м. н., профессор Терехов А.Н.

Научный руководитель:
аспирант кафедры системного программирования Вахитов А.Т.

Рецензент:
аспирант кафедры системного программирования Гуревич Л.С.

Санкт-Петербург
2010

Содержание

1 Введение

Задача идентификации объектов по некоторому набору признаков встречается в жизни достаточно часто; например, при выявлении причин поломки автомобиля, при постановке предварительного диагноза по ряду симптомов, при определении вида животного или растения.

В общем виде эту задачу можно сформулировать так: имеется ряд признаков, каждый из которых может принимать конечное число состояний, имеется набор классов, каждый из которых описывает некоторое количество схожих между собой объектов с помощью их характеристик (то есть класс представляется как набор состояний, которые могут принимать признаки описываемых объектов), и по запросу, содержащему определенные исследователем состояния признаков (не обязательно всех), выявить, каким классам соответствует наблюдаемый объект.

Нередко при решении задачи идентификации мы имеем дело с большим количеством классов и еще большим (до нескольких сотен) количеством признаков. Определение состояний всех признаков при таком их количестве является для исследователя крайне трудоемкой задачей, поэтому важно предоставлять некоторый ранжированный по значимости список признаков, в порядке которого можно наиболее быстро определить класс объекта.

В последнее время к определителям биологических объектов проявляется все больший интерес как к способу развлечения. Например, одним из популярных приложений для Apple iPhone является программа, позволяющая определить птицу по ряду признаков вроде цвета или шаблона полета [1]. Нередко наблюдение за птицами является хобби, и, разумеется, наблюдатель желает знать, какую именно птицу он наблюдает.

При этом кто-то может захотеть использовать приложение-определитель на ноутбуке, кто-то на КПК или веб-сайте. В таком случае, удачным вариантом реализации определителя будет клиент-серверный подход, при котором клиентское приложение общается с веб-сервисом посредством определенного API. Это позволит более гибко подойти к разработке клиента и покрыть большое количество платформ.

Также нельзя не отметить, что в наше время идентификация биологических объектов по ДНК становится все более доступной, но, тем не менее, тем же любителям птиц вряд ли будет доступен подобный подход, плюс, как было отмечено ранее, имеются родственные задачи из других областей, что позволяет нам говорить об актуальности решения данной задачи и в будущем.

2 Постановка задачи

Задача данной работы состоит в построении прототипа универсальной платформы для идентификации биологического объекта по некоторому набору признаков на примере баз данных жесткокрылых насекомых, предоставленных Зоологическим институтом Российской академии наук.

- На основании результатов анализа существующих средств идентификации в биологии выявить их общие недостатки.
- Разработать метод оптимизации определения принадлежности биологического объекта некоторому классу и сделать сравнение с наиболее часто употребляемым на данный момент методом.
- Спроектировать и разработать веб-сервис, на базе которого может быть построен произвольный определитель.
- Разработать демонстрационное клиентское приложение.

3 Обзор предметной области

В этой главе будут рассмотрены наиболее популярные сейчас определители, существующие у них проблемы, способы их исправления, а также общие требования к приложениям для идентификации.

Определители таксономической принадлежности биологических объектов, ориентированные на персональные компьютеры, впервые появились в 1990-х годах. Обзор наиболее популярных, созданных с тех пор приложений, представлен в работе [2]. Как правило, каждая из этих систем ориентирована на определение объектов только конкретных семейств (например, орхидей или грибов-зигомицетов).

3.1 Общие проблемы существующего ПО для идентификации в биологии

В целом представленные программные продукты обладают схожими функциональными возможностями, однако к их безусловным недостаткам можно в первую очередь отнести:

- **Невозможность построения собственного определителя на базе предоставленной системы**
  Исследователь, как правило, не может внести изменения в предлагаемый определитель для поддержания его актуальности на должном уровне, что ограничивает итеративный подход к его построению. Также в случае использования в системе удачного способа оптимизации процесса идентификации, исследователю не предоставляется никакой возможности использовать эту систему, либо ее компоненты, для разработки нового определителя схожей специфики.

- **Жесткая привязка к интерфейсу пользователя**
  Невозможность расширения или улучшения существующего клиентского приложения сторонними разработчиками является серьезным недостатком в случае определителей, ориентированных на использование непрофессионалами.

- **Платформенные ограничения**
  Большинство определителей работают только в операционных системах семейства Windows (INTKEY, IdentifyIt, MEKA, PICKEY), однако есть ряд web-ориентированных решений (Interactive Key to Katydids, NaviKey), часть из которых построена на базе Java-апплетов, что ограничивает возможность их использования на мобильных телефонах и, например, некоторых КПК.

- **Отсутствие API для доступа к функциям определителя**
  Существующие решения предоставляются только в виде программ для конечного пользователя и не включают в себя библиотеки, которые могли бы быть переиспользованы разработчиками других определителей.

3.2 Анализ требований к определителю

Для построения удачного определителя необходимо проанализировать типичный сценарий идентификации. Обычно он состоит из следующих этапов:

- **Вывод пользователю ранжированного списка признаков**
  На этом этапе важно предоставить ту последовательность признаков, которая наиболее быстро и надежно позволит исследователю идентифицировать наблюдаемый объект.

- **Определение пользователем текущих состояний одного или нескольких признаков**

При этом важно предоставить пользователю возможность ввода признаков в произвольном порядке, а также ввода неточных данных (например, нескольких подозрительных состояний вместо одного).

- **Анализ ввода, ранжирование классов по мере соответствия их запросу, ранжирование набора еще не определенных признаков**
  Одним из основных критериев на этом этапе является время обработки запроса. Оно должно оставаться разумным даже для больших баз классов и признаков, обеспечивая непрерывную и комфортную работу исследователя. При ранжировании классов, соответствующих наблюдаемому объекту, важно учесть возможность использования метрики, параметрами которой могут быть вероятность ошибки пользователя при определении введенных признаков, их надежность, а также другие специфические для конкретной области показатели.

- **Вывод пользователю нового списка признаков и подходящих под запрос классов**

3.3 Основные сложности процесса идентификации и пути их оптимизации

К основным характеристикам процесса поиска подходящих наблюдаемому объекту классов, требующим оптимизации в первую очередь, можно отнести длину пути поиска (количество шагов идентификации, предпринимаемых исследователем для выявления наиболее вероятного класса; например, количество определенных им признаков), а также надежность определения.

- **Минимизация средней длины пути определения**
  Мы можем предлагать такой порядок признаков, при котором средние длины путей, приводящих к наиболее подходящим классам, минимальны. Однако не исключены случаи, когда исследователь будет наблюдать

редкий объект, признаки, отделяющие который от остальных, имеют низкий ранг. Поэтому, помимо применения математически обоснованных методов для ранжирования признаков, важно учитывать и экспертные оценки значимости.

- **Повышение надежности определения**
  Например, особенно сложные для идентификации признаки (вероятность ошибки в определении которых высока), но сильно влияющие на конечный результат, следует ранжировать ниже более надежных. Данные для подобного улучшения надежности определения могут быть либо предоставлены экспертом, либо получены в результате наблюдения за рядом успешных определений. Такими статистическими данными могут быть:

  - удачные последовательности признаков, которые привели исследователя к результату, но отличающиеся от предложенной изначально (тут можно учитывать признаки, которые исследователь пропустил);

  - случаи удачного определения, но с ошибками в каких-либо признаках (то есть в случае, когда несколько состояний признаков наблюдаемого объекта не подходят под класс, который исследователь посчитал верным).

Таким образом, помимо общих проблем архитектуры существующих на данный момент определителей, нам также необходимо решить задачу оптимизации процесса идентификации объекта по набору заранее определенных признаков.

4 Оптимизация процесса идентификации

В данной главе рассматривается задача минимизации числа признаков, необходимых для успешного определения класса объекта, а также ее применение в решении задачи оптимального ранжирования признаков в биологических определителях.

Если бы мы для каждого класса могли посчитать минимальную по количеству элементов последовательность признаков, определением которых можно отделить данный класс от всех остальных, то в случае, когда наблюдаемый объект вероятно принадлежит данному классу, мы сможем предоставить исследователю оптимальный набор признаков, чтобы в этом убедиться. Для идентификации же произвольного объекта, ранг признаков, входящих в эти минимальные разделяющие последовательности, должен быть, очевидно, выше.

4.1 Проблема минимального набора признаков отделяющего класс от всех остальных

Для произвольной пары классов a и b обозначим набор признаков, отделяющих a от b как $c_1^{ab}, \dots, c_{k_{ab}}^{ab}$. Пусть всего у нас N классов. Рассмотрим булевскую формулу, значениями переменных c_m^{ij} которой являются *истина* в случае, если этот признак определен исследователем, и *ложь* в противном:

$$(c_1^{a1} \vee c_2^{a1} \vee \dots \vee c_{k_{a1}}^{a1}) \wedge (c_1^{a2} \vee c_2^{a2} \vee \dots \vee c_{k_{a2}}^{a2}) \wedge \dots \wedge (c_1^{aN} \vee c_2^{aN} \vee \dots \vee c_{k_{aN}}^{aN}) \quad (1)$$

Эта формула принимает значения *истина*, если при подстановке в переменные значений, соответствующих выбранным признакам, класс a отделим этими признаками от всех остальных. Отметим, что формула находится в конъюнктивной нормальной форме и не имеет отрицаний.

Таким образом, наша задача заключается в поиске одного или нескольких

минимальных по числу элементов наборов переменных, при присваивании которым значения *истина*, формула также принимает значение *истина*. Решив эту задачу для каждого класса, мы получим необходимые для ранжирующей метрики данные.

Рассмотрим варианты решения.

4.1.1 Преобразование КНФ в ДНФ

Легко заметить, что необходимый набор переменных может быть найден преобразованием данной формулы в дизъюнктивную нормальную форму [3]. Тогда ответом будут являться минимальные по числу переменных конъюнкты.

Эта задача NP-полна, решение ее для больших формул является крайне трудоемким, и вряд ли может быть использовано в нашем случае, поскольку подобные вычисления придется проводить довольно часто и на больших объемах данных.

4.1.2 Использование нейросети

В работе [5] предлагается использовать нейронную сеть для ранжирования признаков. На вход сети подается набор вероятностей того, что наблюдаемый объект принадлежит одному из классов, на выход сеть выдает ранги признаков, на основании которых строится предлагаемый исследователю набор. Обучать сеть предлагается на малых наборах таксонов, для которых можно решить задачу оптимальным способом.

4.1.3 Алгоритм на основе BDD

В ходе работы нами был также предложен метод, использующий BDD (binary decision diagram) [6] для поиска необходимого набора переменных. На рисунке 1 изображена BDD для формулы $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6) \vee (x_7 \wedge x_8)$.

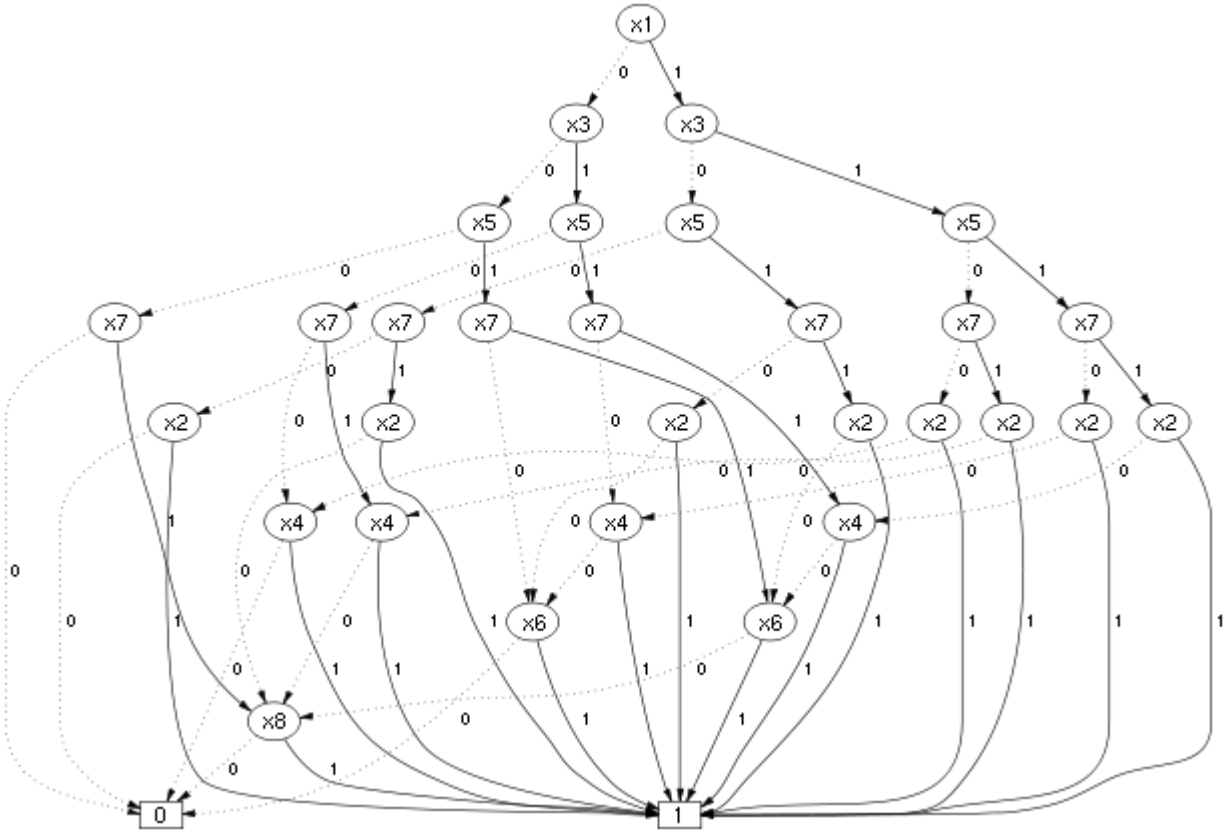

Рис. 1. BDD для формулы $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6) \vee (x_7 \wedge x_8)$

Наша задача может быть решена нахождением минимального по весу пути из узла «1» в корень, при котором весами дуг являются значения переменных, находящихся в начале дуги.

Отметим, что при работе подобного алгоритма, часть времени уйдет на построение BDD и, возможно, ее упрощение. Мы не реализовывали данный подход в силу того, что было найдено решение, которое, с учетом специфики нашей задачи, ожидаемо даст лучшие результаты.

4.1.4 Алгоритм для формул с большим числом удовлетворяющих присваиваний

В ходе работы было отмечено важное свойство, которым обладают формулы вида (1), построенные на основе тестовых баз данных жесткокрылых насекомых, – количество присваиваний, обращающих формулу в истину,

составляет в среднем около 73% от всех возможных присваиваний (коих 2^n, где n – число переменных). Для подсчета был использован рандомизированный алгоритм, проверяющий не все 2^n вариантов, а только 10% из них. Мы можем предположить, что подобные значения будут получены и для других определителей, в которых у каждых двух классов достаточно велико количество разделяющих их признаков.

Это наблюдение позволяет нам использовать для решения задачи подход, наиболее соответствующий ее специфике, а именно алгоритм для формул с большим количеством удовлетворяющих присваиваний [7]. Алгоритм возвращает наименьшее по числу элементов присваивание (включая и присваивание переменным значений ложь). Нам же необходима минимизация по числу присваиваний значений истина, а также возможность получения нескольких наименьших присваиваний, поэтому мы будем использовать данный алгоритм с некоторой модификацией.

Пусть дана КНФ-формула f с переменными $a_1, a_2, \dots, a_n$ и K – число наименьших присваиваний, которое нужно получить.

0) $i := 0; \Theta := \varnothing; \Phi_0 := \{f\}; for\ m \in \{1, \dots, n\}\ \Phi_m := \varnothing;$

1) В каждой $f_{i,j} \in \Phi_i = \{f_{i,1}, f_{i,2}, \dots, f_{i,q_i}\}$ выбираем наименьший по числу элементов дизъюнкт $l_1 \vee l_2 \vee \dots \vee l_{s(i,j)}$

2) Для каждой $f_{i,j}$ получаем на ее основе новые формулы $g_{i,j,1}, g_{i,j,2}, \dots, g_{i,j,s(i,j)}$ с помощью следующих присваиваний: $\{l_1\}, \{\bar{l}_1, l_2\}, \dots, \{\bar{l}_1, \bar{l}_2, \bar{l}_3, \dots, l_{s(i,j)}\}$, где l_n означает присваивание переменной l_n значения *истина*, а $\bar{l}_n$ означает присваивание переменной l_n значения *ложь*. Для всех формул, значение которых *истина*, добавляем $t(g_{i,j,m})$ в Θ, где $t(g_{i,j,m})$ – множество присваиваний значений *истина*, которыми из f была получена эта формула. Также проверяем, получили ли мы K

присваиваний и возвращаем Θ, если это так.

3) Для всех формул $g_{i,j,m}$, значение которых не определено (то есть не *истина* и не *ложь*) добавляем их в $\Phi_{|t(g_{i,j,m})|}$.

4) $i := i+1;$ Если $i > n$, возвращаем Θ, в противном случае переходим на шаг 1.

Данный алгоритм был реализован на языке Java в виде библиотеки классов, которую впоследствии будет использовать веб-сервис для ранжирования признаков.

Класс `Algorithm` библиотеки `com.webkey.minsat` реализует функцию `calculate` для подсчета заданного количества минимальных по числу элементов удовлетворяющих присваиваний:

```
public static ArrayList<ArrayList<Variable>>
calculate(CnfFormula initialFormula, int number)
```

Все классы полностью документированы в Javadoc [8], скомпилированная документация предоставляется вместе с исходным кодом системы.

4.2 Ранжирование признаков

Основная наша цель – это ранжирование признаков с целью сокращения пути поиска класса, соответствующего наблюдаемом объекту. Для этого нами была решена задача нахождения минимального набора признаков, отделяющего один класс от всех остальны. Как было отмечено ранее, признаки, входящие в такие наборы, должны ранжироваться выше остальных. Используем эту идею для построения метрики.

4.2.1 Реализация

Для случая, когда статистические данные еще не накоплены, и мы не имеем никаких экспертных оценок признаков, используем самый простой вариант метрики. В будущем, по мере появления новых факторов ранжирования, метрика может быть модифицирована.

Поскольку признаки в нашем определителе могут иметь произвольное количество состояний, мы в некоторых случаях не сможем говорить о том, разделяет ли данный признак два класса (это случаи, когда множества состояний этого признака, присущих классам, пересекаются и не содержатся один в другом), поэтому будем рассматривать минимальные множества *состояний*, разделяющих классы.

Рассмотрим определитель, каждый признак в котором имеет только два состояния: признак *присутствует* у наблюдаемого объекта и признак *отсутствует*. Мы можем найти K минимальных по числу элементов наборов $\{s_k\}^{i,j}_{j\in[1,n]}$ признаков (состояний в изначальной постановке), отделяющих класс i от остальных. Тогда ранг произвольного признака (в изначальной постановке имеющего несколько состояний) может быть посчитан следующим образом:

$$rate(c)=\sum_{s\in states(c)}\ \sum_{i\in[1,N],\,j\in[1,K]}\left\{P_{i,j}(s)\middle|s\in\{s_k\}^{i,j}_{j\in[1,K]}\right\}$$

$$P_{i,j}(s)=\frac{1}{\left|\{s_k\}^{i,j}_{j\in[1,K]}\right|}$$

В качестве функции $P_{i,j}(s)$ можно использовать отличную от предложенной (например, содержащую экспертные оценки состояния или признака).

Чем выше ранг признака, тем ценнее он в диагностическом плане для исследователя.

4.2.2 Сравнение с энтропийным методом

Традиционно в биологических определителях используется метод на основании расчета энтропии распределения возможных состояний для каждого признака [9].

Признаки сортируются в порядке убывания энтропии Шеннона:

$H(c) = -\sum_{i=1}^{n_i} |\chi_c(s_i)| \ln |\chi_c(s_i)|$, где $\chi_c(s_i)$ – множество классов, удовлетворяющих состоянию s_i признака c.

Для сравнения предложенного нами метода с энтропийным методом, мы считали среднее число шагов для определения объекта каждого класса (то есть до момента, когда остается только один класс, подходящий под введенные данные) в порядке ранжированного списка признаков. На трех различных определителях мы получили следующие результаты (где K – количество минимальных путей при подсчете для одного класса):

- **35 классов, 38 признаков, 114 состояний**
  Энтропийный метод: 7.71 шагов
  Метод минимальных разделяющих множеств:

K = 1	**K = 7**	**K = 15**
10.5	9.54	6.98

- **130 классов, 24 признака, 103 состояния**
  Энтропийный метод: 14.52 шага
  Метод минимальных разделяющих множеств:

K = 1	**K = 7**	**K = 15**
11.63	11.28	11.24

- **1039 классов, 331 признак, 909 состояний**
  К сожалению, у нас не было достаточно вычислительных мощностей для того, чтобы за разумное время провести предлагаемые тесты. Был получен только один результат для K = 1, среднее считалось не по всем 1039 классам, а по случайным 100.
  Энтропийный метод: **53.17** шагов
  Метод минимальных разделяющих множеств: **24.3** шага

По полученным результатам мы можем предположить, что предложенный нами метод с ростом числа классов и возможных признаков и состояний дает ощутимо лучшие результаты нежели энтропийный.

Следует отметить, что при тестировании определителя с ранжированием признаков по методу минимальных разделяющих множеств, ранги признаков считаются только один раз и не пересчитываются после каждого шага определения, в отличии от энтропии, подсчитываемой на каждом шаге. Это, безусловно, поможет сэкономить часть ресурсов веб-сервиса без уменьшения качества определителя.

4.3 Ранжирование объектов

На данный момент не используется какой-либо специальной метрики для ранжирования объектов. Объект считается соответствующим классу, если множество состояний его признаков пересекается с множеством состояний признаков класса. Ранг класса относительно наблюдаемого объекта тем выше, чем больше элементов содержит пересечение множеств их состояний.

4.4 Рекомендации по применению метода

- При начальном ранжировании признаков рекомендуется проводить ряд тестов для выявления оптимального значения параметра **K**, при котором

среднее число шагов идентификации будет минимальным. Особенно важной эта рекомендация является для небольших определителей, число классов в которых не превышает 100.

- По мере накопления статистики по результатам определения, рекомендуется скорректировать значение параметра **K** таким образом, чтобы при описанном тестировании предпочтение отдавалось сокращению пути поиска для наиболее часто встречающихся объектов.

5 Реализация универсального определителя

В данной главе рассматривается решение для построения универсального определителя, лишенного основных недостатков существующих систем.

5.1 Решение общих проблем существующего ПО

Как было отмечено ранее, у существующих до настоящего времени определителей есть ряд общих проблем: невозможность построения собственного определителя на базе предоставленной системы, жесткая привязка к интерфейсу пользователя, платформенные ограничения, отсутствие API для доступа к функциям определителя. Клиент-серверная архитектура позволяет решить сразу несколько этих проблем следующим образом:

- **Сервер**
  Хранит данные по зарегистрированным определителям, отвечает за ранжирование признаков и классов в процессе определения, собирает статистику. В API веб-сервиса могут быть добавлены функции регистрации нового определителя и изменения существующих данных.

- **Клиент**
  Наличие общего API для доступа к функциям сервера делает возможным построение клиентского приложения на любой платформе и с любым пользовательским интерфейсом.

Также немаловажным принципом построения демонстрационного приложения является возможность запуска разработанного веб-сервиса на любой популярной в настоящее время операционной системе, будь то *nix, Windows или MacOS.

5.2 Общая архитектура приложения

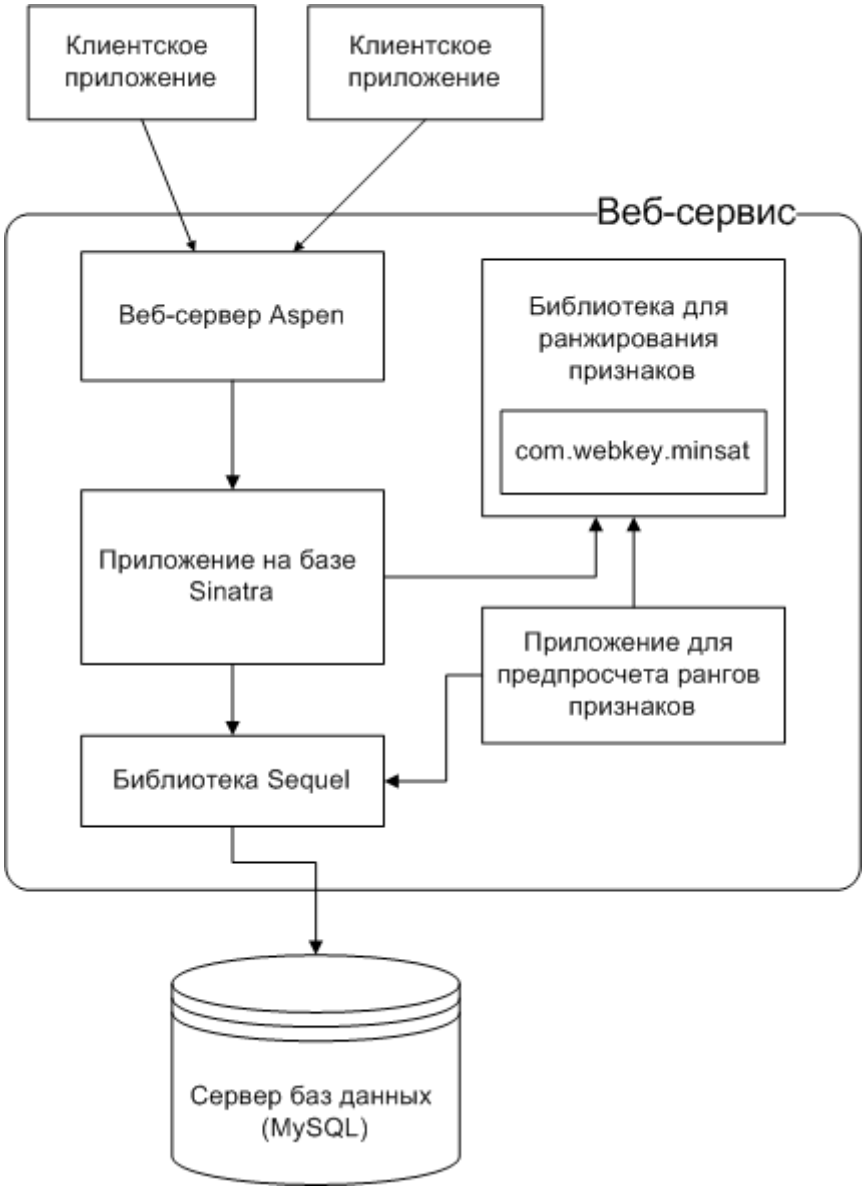

Рис. 2. Архитектура приложения

На рисунке 2 представлены основные компоненты системы:

- Ряд клиентских приложений, отвечающих за передачу пользователю информации, полученной от сервера. Демонстрационный клиент представляет собой сайт, работающий на отдельном сервере. Реализован на популярном MVC фреймворке – Zend Framework [10].

- Веб-сервис, постоенный на базе сервера приложений Aspen [12],

написанного на JRuby [13], непосредственно самого приложения разработанного на фреймворке Sinatra [14].

- База данных под управлением сервера MySQL [15]. Для доступа к данным была использована ORM библиотека Sequel [16].

- Демон предпросчета рангов признаков для вновь добавленных определителей.

5.3 Реализация веб-сервиса

При выборе платформы для реализации веб-сервиса мы опирались на следующие факторы:

- **Скорость разработки**
  Поскольку задачей является реализация именно прототипа, демонстрирующего принципы архитектуры в целом, скорость разработки сервиса и уровень абстракции являются одними из наиболее важных факторов.

- **Доступность технологии для всех платформ**
  Кросс-платформенность как клиентской части, так и серверной, является одним из важных критериев успешности разрабатываемого приложения. Независимость в этом плане позволяет расширить круг возможных пользователей системы в целом, а также позволяет, не ограничиваясь рамками одной платформы, выбирать инструменты для расширения функциональности веб-сервиса.

- **Использование быстрых языков**
  В случае реализации веб-сервиса на скриптовом языке, необходима возможность написания компонентов чувствительных к времени обработки или ресурсам сервера, на языке, обладающем большей

скоростью (например, C/C++ или Java).

- **Библиотека готовых решений**
  Наличие фреймворков для веб-разработки, доступа к данным, передачи сообщений по популярным протоколам также является важным критерием, способствующим ускорению разработки и удешевлению поддержки разрабатываемой системы.

Всеми этими достоинствами обладает одна из реализаций языка Ruby – JRuby, написанная на Java. Платформа позволяет использовать любые Java-библиотеки, и в то же время обладает богатым выбором готовых решений для веб-разработки.

Одним из таких фреймворков является Sinatra, позволяющий быстро строить системы, основанные на архитектурном подходе для распределенных приложений REST [17], [18]. В идеологию подхода REST отлично вписываются методы, предоставляемые API разрабатываемого веб-сервиса.

REST подразумевает использование протокола HTTP [19] для передачи сообщений между сервером и клиентом. При этом данные обычно передаются в JSON [20], XML [21] или YAML [22] формате. В разрабатываемом приложении используется формат JSON, который позволяет без какой-либо конвертации создавать объекты языка JavaScript, а также с помощью встроенных функций языка сериализовать объект в JSON-строку [23] (клиентское приложение активно использует технологию AJAX [24], поэтому удобство работы с форматом в языке JavaScript является важным критерием отбора).

Для упрощения развертывания системы в произвольном окружении, в качестве сервера приложений был выбран Aspen. В нашем случае для запуска веб-сервиса необходимо только запустить исполняемый файл (естественно, после удовлетворения зависимостей от сторонних библиотек).

5.3.1 Схема базы данных

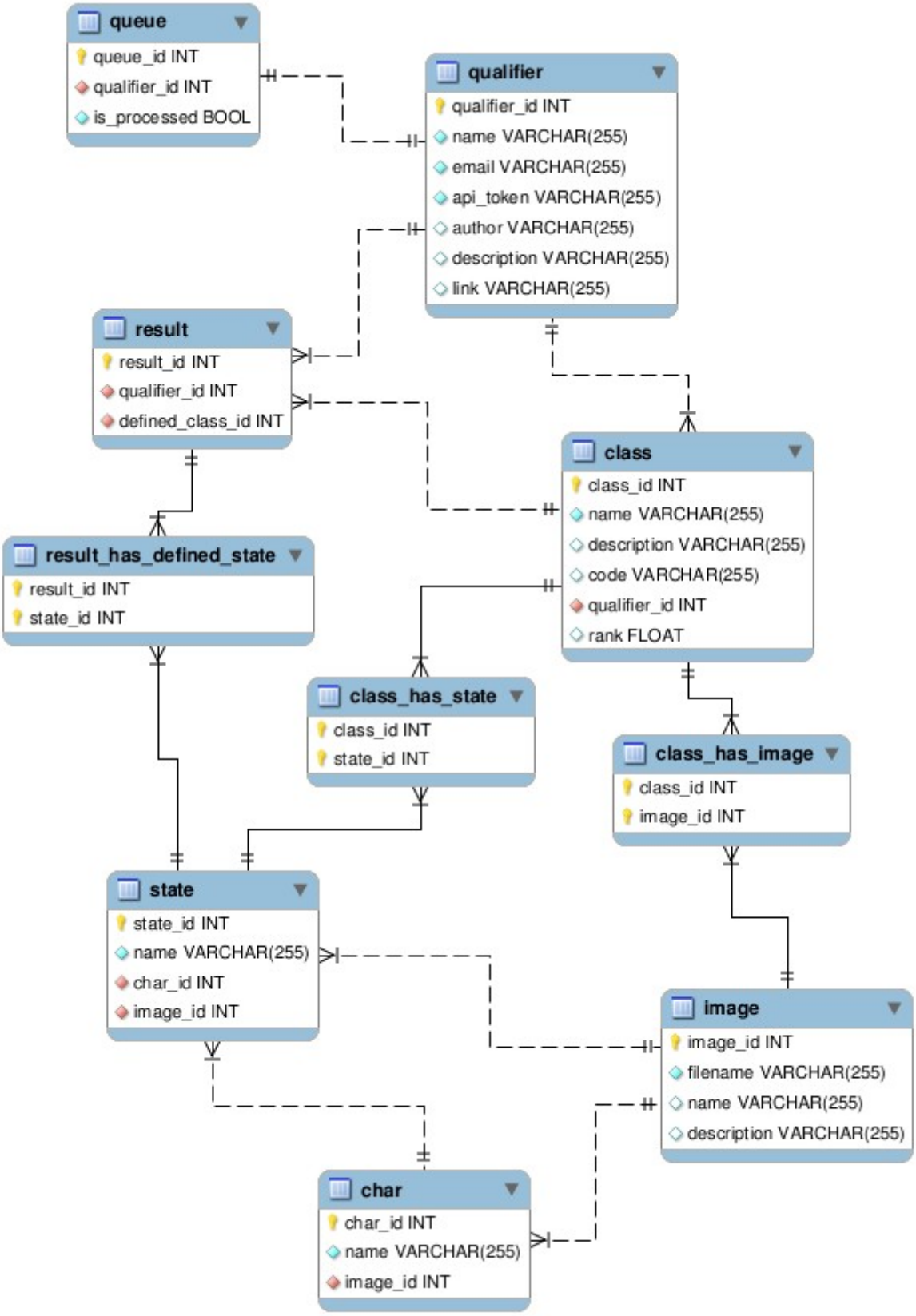

Рис. 3. Схема базы данных веб-сервиса

Продемонстрированная на рисунке 3 база данных спроектирована с учетом того, что веб-сервис будет обслуживать произвольное число определителей.

- Таблица **qualifier** отвечает за представление сущности определителя. В таблице хранятся общие данные (название, имя автора, адрес его электронной почты, ссылка на сайт, описание), а также ключ доступа к данному определителю (*api_token*) через API.

- Каждый определитель имеет множество идентифицируемых им классов – таблица **class**, которая помимо имени класса хранит его описание, а также произвольный ключ, который может быть использован, например, для связи классов клиентского приложения с классами, хранящимися на сервере. У каждого класса может быть одно или несколько изображений иллюстрирующих, например, объекты данного класса.

- Признаки хранятся в таблице **char**. У каждого признака есть имя, а также иллюстрация.

- Таблица **state** отвечает за представление состояний. Каждое состояние принадлежит только одному признаку, а связь классов и состояний обеспечивается таблицей **class_has_state**.

- Помимо данных, относящихся непосредственно к определителям, в базе также хранится и некоторая статистическая информация. В таблице **result** собираются все успешные пути поиска, а также классы, к которым они привели. В перспективе эта информация может быть использована в качестве одного из критериев ранжирования признаков.

- Таблица **queue** используется для реализации очереди обработки вновь зарегистрированных определителей.

База данных работает под управлением сервера MySQL, который был выбран

как один из наиболее распространенных и надежных, а также поддерживаемый большим количеством готовых библиотек для доступа к данным.

В качестве такой библиотеки нами была использована Sequel, как достаточно нетребовательная к ресурсам сервера, и в то же время обладающая большим набором функциональных возможностей.

5.3.2 Описание API

Клиент посылает запросы серверу по HTTP протоколу на определенный URL, и в зависимости от типа запроса (GET, PUT, POST, DELETE – указывается в скобках), получает необходимые данные, либо сообщение об успехе или ошибке во время выполнения операции.

URL методов для получения каких-либо данных из определителя, либо методов для запроса подходящих классов и рангов признаков формируется из префикса с идентифицирующей определитель строкой и имени метода. Там где необходимо, этот префикс отмечен как **:api_token**.

/register *(POST)*

Создает новый определитель. В запросе необходимо передать следующие параметры (необязательные взяты в квадратные скобки):

- **name** – название определителя.
- [**author**] – имя автора или название организации, составившей определитель.
- [**link**] – сайт автора.
- [**description**] – описание определителя.
- [**api_token**] – желаемая идентифицирующая определитель строка. Если не

указано, строка формируется как уникальная последовательность из нескольких случайных символов.

- **chars** – все признаки определяемых классов в формате JSON в виде массива объектов с полями: *id* (уникальный номер признака), *name* (имя признака), *image* (ссылка на изображение).

- **states** – состояния признаков. JSON массив объектов с полями: *id* (уникальный номер состояния), *name* (имя состояния), *char_id* (id признака, к которому относится данное состояние), *image* (ссылка на изображение).

- **classes** – классы, идентифицируемые определителем. JSON массив объектов с полями: *name* (название класса), *description* (описание класса, разрешена HTML разметка), *code* (код, который будет возвращаться с информацией о классе), *states* (массив id состояний, соответствующих классу), *images* (массив ссылок на изображения).

/:api_token/chars (GET)

Возвращает все признаки в виде JSON массива объектов с полями: *id*, *name*, *image*, *rank* (ранг данного признака; чем выше ранг, там ценнее признак в диагностическом плане).

/:api_token/char/:id (GET)

Информация о признаке по id. Возвращает JSON объект с полями: *id*, *name*, *image*, *rank.*

/:api_token/classes (GET)

Возвращает все классы в виде JSON массива объектов с полями: *id*, *name,*

description, *code*, *states*, *images*.

/:api_token/class/:id (GET)

Информация о классе по id. Возвращает JSON объект с полями: *id*, *name, description, code, states, images*.

/:api_token/query/:query_string (GET)

Выполняет запрос на определение подходящего класса. **:query_string** – строка, содержащая выбранные исследователем состояния признаков, разделенные символом /. Метод возвращает JSON объект с полями: *classes* (массив объектов, хранящих *id* класса и *rank*), *chars* (массив объектов, хранящих *id* признака и *rank*).

/:api_token/result (POST)

Регистрирует результат идентификации. В запросе необходимо передать следующие параметры:

- **defined_class_id** – id класса, который исследователь посчитал наиболее соответствующим своему запросу.
- **query** – JSON массив id состояний, определенных исследователем.

5.3.3 Подсчет ранжировки признаков

Процесс регистрации нового определителя потенциально может занимать заметное для пользователя количество времени даже в случае небольших по объему данных. Поэтому подсчет изначальной ранжировки признаков происходит не сразу, а добавляется в очередь на обработку. Тем самым мы даем пользователю ответ о добавлении его данных незамедлительно, но также предупреждаем о том, что созданный им определитель станет доступен для

запросов через некоторое время, о чем он будет оповещен по электронной почте.

Раз в час запускается скрипт, подсчитывающий ранги признаков вновь добавленных определителей. В будущем подобный скрипт может также пересчитывать ранги существующих признаков с учетом собранной статистики.

5.4 Реализация клиента

Для демонстрации работы веб-сервиса было реализовано простейшее клиентское приложение, позволяющее пользователю идентифицировать наблюдаемый объект.

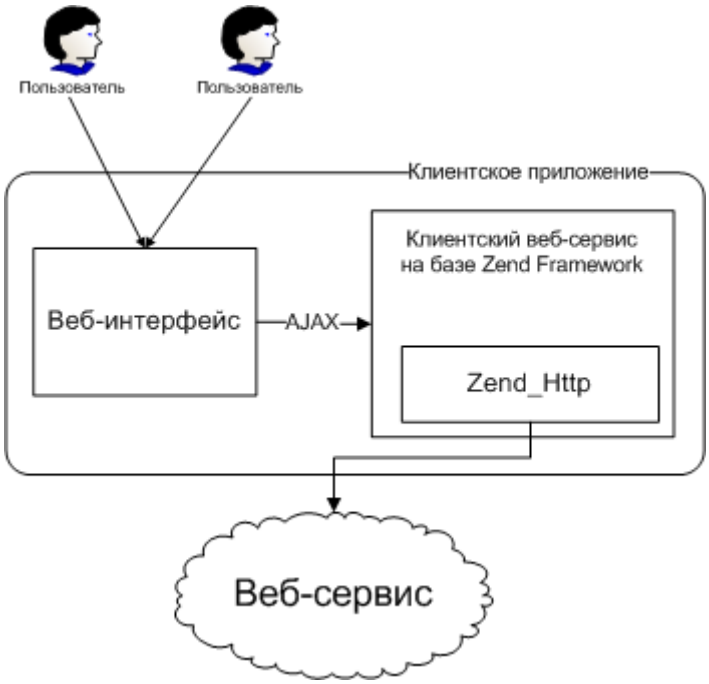

Рис. 4. Архитектура клиентского приложения

Суть работы клиента заключается в следующем:

- Точкой входа для пользователя является статическая веб-страница, со встроенным в нее JavaScript кодом, отвечающим за обновление данных на странице, а также за отправку запросов с введенными пользователем признаками наблюдаемого объекта.

Поскольку AJAX-запросы разрешены только в пределах одного домена (то есть запросы со некоторого сайта могут быть отправлены только к страницам того же сайта, что является частью политики безопасности любого веб-браузера), JavaScript сценарий отправляет запросы к специальному клиентскому веб-сервису, расположенному на том же сервере, являющемуся, по-сути, зеркалом внешнего веб-сервиса.

- Клиентский веб-сервис посредством библиотеки Zend_Http [11] формирует HTTP запрос к внешнему веб-сервису, и, получив от него ответ, без изменений возвращает его.

- Страница, на которой находится пользователь, получает ответ от клиентского веб-сервиса, интерпретирует его и без перезагрузки страницы обновляет отображаемые данные.

Клиентское приложение написано с применением библиотек Zend Framework и шаблона проектирования MVC [25].

Следует отметить, что для повышения удобства работы исследователя, клиент может быть расширен следующими функциями:

- Возможностью отката процесса определения на один или несколько шагов назад.

- Возможностью получения полных сведений о любом классе или признаке независимо от шага процесса определения.

- Добавлением в интерфейс пользователя изображений классов, признаков и состояний.

- Возможностью отправки результата идентификации (успешной или нет).

6 Заключение

6.1 Достигнутые результаты

- Проведен анализ проблем существующих биологических определителей, а также выявлены их основные недостатки.
- Найден метод оптимизации процесса идентификации объекта по произвольному числу признаков, куда входит:
  - метод решения задачи о минимальном по числу элементов разделяющем два произвольных класса множестве признаков;
  - метод ранжирования признаков для минимизации числа определяемых признаков.
- Проведено имитационное моделирование, которое показало, что разработанный метод ускоряет определение объекта по среднему числу используемых признаков на 20-50% по сравнению с традиционно используемым энтропийным методом.
- Даны рекомендации по применению нового метода в биологических определителях.
- Реализован веб-сервис, лишенный недостатков текущих существующих решений.
- Реализовано демонстрационное клиентское приложение.

6.2 Направления развития

В качестве направлений развития системы можно выделить:

- **Дальнейшую оптимизацию процесса идентификации объектов**

Сюда относится использование данных накопленной по результатам определений статистики, что позволит задать более точную, с точки зрения минимизации определяемых признаков, метрику.

Также возможен пересчет рангов признаков в зависимости от определенных пользователем состояний. Это позволит системе оптимизировать поиск даже редких классов.

Немаловажным фактором улучшения метрики ранжирования признаков является добавление в ее параметры экспертной оценки надежности признака и вероятности его ошибочного определения.

- **Улучшение пользовательского интерфейса клиентского приложения**
  В первую очередь это возможность отката на произвольный шаг идентификации, а также использование определителя как справочника по его классам и их признакам.

7 Список литературы

[1] Приложение iBird Explorer PRO
// http://itunes.apple.com/app/ibird-explorer-pro/id308018823?mt=8

[2] А.Л. Лобанов, А.Г. Кирейчук, И.С. Смирнов. Биологическая диагностика: история, современное состояние, проблемы, 2009.
// http://www.zin.ru/projects/WebKey-X/basis.htm

[3] Peter Bro Miltersen , Jaikumar Radhakrishnan , Ingo Wegener . On converting CNF to DNF . Basic Research in Computer Science , December 2003.

[4] Marco Budinich . Neural Networks for NP-complete Problems . Invited talk at World Congrress on Nonlinear Analysis - 23-27 July 1996 - Athens, Greece .

[5] А.Т. Вахитов, О.Н. Граничин, А.Г. Кирейчук, А.Л. Лобанов. Параллельный алгоритм обучения для интерактивного политомического определителя биологических видов, 2009.

[6] H.R. Andersen. An Introduction to Binary Decision Diagrams. Lecture Notes. IT University of Copenhagen. 1999.

[7] E.A. Hirsch. A Fast Deterministic Algorithm for Formulas That Have Many Satisfying Assignments. L. J. of the IGPL, Vol. 6 No. 1, pp. 59-71, 1998.

[8] Javadoc 1.4.2 Tool Reference Page.
// http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/index.html

[9] А.В. Свиридов. Ключи в биологической систематике: теория и практика. М.:Издательство Московского Университета. 1994. 224 с.

[10] Zend Framework Programmer's Reference Guide.
// http://zendframework.com/manual/en

[11] Документация по компоненту Zend_Http.
// http://zendframework.com/manual/en/zend.http.html

[12] Git-репозиторий проекта веб-сервера Aspen.
// http://github.com/kevwil/aspen

[13] Язык JRuby. // http://jruby.org

[14] Sinatra framework Documentation.
// http://www.sinatrarb.com/documentation

[15] MySQL Documentation. // http://dev.mysql.com/doc/index.html

[16] Documentation for Sequel. (v3.11.0)
// http://sequel.rubyforge.org/documentation.html

[17] Roy Fielding. Architectural Styles and the Design of Network-based Software Architectures. 2000. pp. 76-106.

[18] Cesare Pautasso, Olaf Zimmerman, Frank Leymann. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. 17th International World Wide Web Conference, 2008.

[19] Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. Revision: 1.8. 2004
// http://www.w3.org/Protocols/rfc2616/rfc2616.html

[20] The application/json Media Type for JavaScript Object Notation (JSON). RFC 4672. July 2006.
// http://www.ietf.org/rfc/rfc4627.txt?number=4627

[21] Extensible Markup Language (XML). // http://www.w3.org/XML/

[22] YAML Ain’t Markup Language (YAML). Version 1.2. // http://www.yaml.org/

[23] JSON in JavaScript. // http://www.json.org/js.html

[24] Jesse James Garrett. Ajax: A New Approach to Web Applications. 2005. // http://www.adaptivepath.com/ideas/essays/archives/000385.php

[25] Derek Greer. Interactive Application Architecture Patterns. // http://www.aspiringcraftsman.com/2007/08/interactive-application-architecture/

[26] А.Т. Вахитов , О.А. Граничина . Алгоритмы классификации за минимальное число шагов, 2006.